RANsacked: A Domain-Informed Approach for Fuzzing LTE and 5G RAN-Core Interfaces*

Nathaniel Bennett
University of Florida
Gainesville, FL, USA
bennett.n@ufl.edu

Weidong Zhu
University of Florida
Gainesville, FL, USA
weidong.zhu@ufl.edu

Benjamin Simon
University of Florida
Gainesville, FL, USA
bsimon4@ufl.edu

Ryon Kennedy
University of Florida
Gainesville, FL, USA
ryonkennedy@ufl.edu

William Enck
North Carolina State University
Raleigh, NC, USA
whenck@ncsu.edu

Patrick Traynor
University of Florida
Gainesville, FL, USA
traynor@ufl.edu

Kevin R. B. Butler
University of Florida
Gainesville, FL, USA
butler@ufl.edu

ABSTRACT

Cellular network infrastructure serves as the backbone of modern mobile wireless communication. As such, cellular cores must be proactively secured against external threats to ensure reliable service. Compromised base station attacks against the core are a rising threat to cellular networks, while user device inputs have long been considered as an attack vector; despite this, few techniques exist to comprehensively test RAN-Core interfaces against malicious input. In this work, we devise a fuzzing framework that performantly fuzzes cellular interfaces accessible from a base station or user device, overcoming several challenges in fuzzing specific to LTE/5G network components. We also introduce ASNFuzzGen, a tool that compiles ASN.1 specifications into structure-aware fuzzing modules, thereby facilitating effective fuzzing exploration of complex cellular protocols. We run fuzzing campaigns against seven open-source and commercial cores and discover 119 vulnerabilities, with 93 CVEs assigned. Our results reveal common implementation mistakes across several cores that lead to vulnerabilities, and the successful coordination of patches for these vulnerabilities across several vendors demonstrates the practical impact ASNFuzzGen has on hardening user-exposed cellular systems.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; **Software and application security**; • **Networks → Mobile networks**.

*This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, http://dx.doi.org/10.1145/3658644.3670320.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '24, October 14–18, 2024, Salt Lake City, UT, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0636-3/24/10…$15.00
https://doi.org/10.1145/3658644.3670320

KEYWORDS

cellular security, fuzzing, rogue base station

ACM Reference Format:
Nathaniel Bennett, Weidong Zhu, Benjamin Simon, Ryon Kennedy, William Enck, Patrick Traynor, and Kevin R. B. Butler. 2024. RANsacked: A Domain-Informed Approach for Fuzzing LTE and 5G RAN-Core Interfaces. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3658644.3670320

1 INTRODUCTION

Cellular telecommunication systems represent critical infrastructure and the primary means by which most of the world communicates. As successive generations of cellular technologies have been deployed, the underlying technologies and communication protocols have become more open and accessible, particularly given the IP-based core network that supports Long Term Evolution (LTE) and 5th-Generation (5G) systems. These new generations of cellular systems bring increasing complexity as well, with many different components and interfaces between them. While past work has considered the security of these interfaces based on standards documents [35], implementations of the cellular core have not been comprehensively assessed for vulnerabilities despite their use in both academic and commercial settings.

Fuzzing has become an increasingly popular and effective means to assess system security. Advances in fuzzers have led to solutions that perform well against large code bases that can carry substantial internal state, but the most effective solutions are *coverage-guided* (i.e., *grey-box*) fuzzers that discover new test cases through mutations based on an initial corpus of inputs. While this approach can be parallelized and lead to very high numbers of executed test cases per second, they encounter challenges when used in the complex cellular core environment, which can be summarized as follows:

1. Fuzzing-Averse Cellular Core Architectural Design. By configuring and performing ground-truth tests on several LTE implementations as testbeds, we observe several characteristics of Radio Access Network (RAN)-Core interfaces that incur additional complexity in applying a fuzzing harness, lead to significant performance degradation, or require external components that would introduce erratic behavior during repeated fuzzing. These characteristics include intentional delaying of transport protocol messages, distributed or

multiprocess architectures for certain component implementations, required interactions across multiple network interfaces to initialize a channel for fuzzing, and external dependencies to other core components to exchange messages during operation. Certain core components additionally incur significant performance penalties due to initialization, further reducing fuzzing speeds.

2. Breadth/Complexity of Control-Plane Messages. Most network protocols define a handful of base message variants and tens or occasionally hundreds of data types within messages. In stark contrast with this, control-plane protocols define nearly 100 distinct base message variants and over 1400 data types, with complex nested data type relations encoded in a dense Abstract Syntax Notation No. 1 (ASN.1) specification. The combined breadth and depth of protocol data types make for an expansive range of inputs that need to be explored to achieve any reasonable measure of protocol coverage. To further complicate matters, the encoding scheme employed by the control plane is susceptible to invalid inputs and experimentally rejected over 95% of fuzz-generated inputs. Taken together, these represent a significant hurdle to fuzz RAN-core interfaces effectively.

To address these challenges, we present a new approach to fuzzing cellular core components, with particular focus on protocol interfaces that receive input directly from mobile handsets and base stations. Recent reports indicate that 18% of cellular security incidents target base station equipment [21], making attacks via such RAN devices a credible threat. We conduct experiments on six different implementations of LTE and 5G cores, several of which are actively used in commercial, industrial and/or and private network contexts. We set up and run these cores to develop fuzzing seeds based on network packet traces, and we develop a *structure-aware* approach (ASNFuzzGen) to fuzzing the S1AP/NGAP protocols that is generalizable to any Packed Encoding Rules (PER) ASN.1 specification, including those seen in other cellular, automotive and space systems. We also perform a black-box fuzzing campaign against a proprietary LTE core based on the domain-informed techniques we develop through our analysis of open-source cores. In total, we discover 119 vulnerabilities across these core implementations, 96 of which have been assigned CVE identifiers. We demonstrate that our domain-informed, structure-aware fuzzing methodology finds vulnerabilities efficiently and scalably, and publicly release ASNFuzzGen to aid further research and ASN.1 fuzzing efforts[1].

2 BACKGROUND

LTE and 5G cores are composed of several networked components of differing roles; together, these components enable a cellular core to scalably serve network traffic to User Equipment (UE) and meter plan usage. Cellular networks can be divided into the RAN, which comprises devices that communicate wirelessly at the edge of the network, and the core, which comprises provider-operated equipment that enables internetworking.

RAN Architecture. Within the RAN, a UE connects via radio to the base station (eNodeB/gNodeB for LTE or 5G respectively). The base station communicates control signalling directly with the core via S1 Application Protocol (S1AP)/Next Generation Application Protocol (NGAP) protocols, and transparently forwards Non-Access Stratum (NAS) control information between the UE and the core (as seen in Figure 1). Control information is handled by the Mobility Management Entity (MME) or Access and Mobility Management Function (AMF) (which we hereafter refer to collectively as the Mobility Management (MM) component) for LTE/5G, respectively. User data is passed to the Serving Gateway (SGW) or Access Gateway (AGW) (which we refer to as the Gateway (GW) component) respectively.

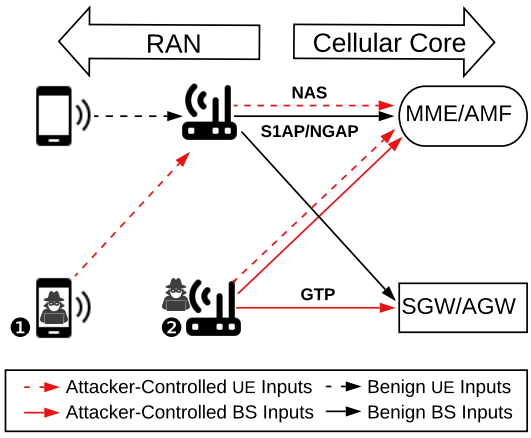


Figure 1: Message flow of potential adversary inputs. A malicious UE (1) may send malformed NAS payloads that are forwarded by a benign base station (BS) over S1AP/NGAP. A malicious BS (2) may send malformed GTP, S1AP/NGAP, and NAS payloads directly to core components.

Control Plane (S1AP/NGAP & NAS). The S1AP/NGAP protocols perform two distinct roles: they enable core MM components to manage base station operation and receive radio service metadata, and they serve as a transport channel over which authentication and handover information is exchanged between the UE and the MM component via NAS messages. NGAP is largely an extension of S1AP with additional functionality for 5G. While the two protocols are not cross-compatible, both handle initial base station registration via a single unauthenticated setup message, and subsequent UE authentication and handover actions involve near-identical message flows.

S1AP and NGAP are both specified with ASN.1, a language for describing data structures that can be serialized/deserialized in a platform-independant way. ASN.1 is widely used not only in the cellular domain, but in space communications, TLS and LDAP authentication, automotive standards, manufacturing/HVAC systems, payment cards, and other industries. While ASN.1 use spans back as far as the 1980's, vulnerabilities in both deserializing ASN.1 payloads and interpreting decoded fields continue to affect the security of communication chanels (as our work will show for the cellular domain).

ASN.1 defines several different methods by which a payload may be serialized/deserialized, known as *encoding rules*. Of these,

[1]https://github.com/FICS/asnfuzzgen

Packed Encoding Rules is designed to minimize the number of bytes used during serialization of a structure. Packed Encoding Rules encodes individual fields in slices of bits, rather than aligning by byte boundaries, and any field values that can be inferred by the recipient are omitted during the serialization process. S1AP and NGAP use Packed Encoding Rules as their encoding method.

The NAS protocol is not defined using ASN.1, but rather follows its own custom data format. NAS handles authentication exchange between the core and the UE, passes configuration parameters to the UE, and enables Short Messaging Services (SMS). It contains a much smaller set of data types and message variants than S1AP/NGAP.

User Plane (GTP). Once a UE has connected and authenticated with a cellular network, the base station exchanges its data and IMS service traffic with the core network via GPRS Tunnelling Protocol (GTP). GTP is a lightweight protocol designed to encapsulate IP packets with a short header containing sender identification, message type and a set of optional extensions. As the encapsulated data is transparently passed through the cellular core without further processing, we do not include GTP as a target for structure-aware fuzzing.

Core Architecture. Within the core, both the MM and the GW components handle traffic from several base stations; it is not uncommon for a single MM component to handle control signalling for an entire metropolitan area. The MM and GW components both handle sensitive user information and must be operational for the cellular core to function. Likewise, both components handle communications at a much wider scale than other user-facing components, such as base stations. As such, they represent significant targets for both denial of service and remote code execution attacks.

3 THREAT MODEL

As mentioned above, the MM and GW components both handle communications for users across potentially dozens or hundreds of base stations spanning a large-scale metropolitan area, and both components must be operable for cellular services to work. For the scope of our paper, we consider two potential goals of an adversary: first, to *deny access to cellular services* across a wide metropolitan area, and second, to *remotely execute code within the network core*. The latter gives an attacker the ability to additionally access sensitive user data or encryption keys, manipulate cellular network behavior, or establish a foothold within the core whereby further attacks could be mounted on more critical components. In carrying out these goals, we assume that the attacker would have the capability to act from one of two threat vectors, as depicted in Figure 1 and described hereafter:

1. Compromised Base Station. Given the network access of a base station, an attacker can exchange control plane (S1AP/NGAP) packets with the MM and user plane packets (GTP) with the GW component. However, the functions that can be carried out using these interfaces are restricted in scope to authentication and information exchange between the UE and the core, providing metadata on a UE to the MM component, and performing handovers of UE devices to other base stations. Base station compromise also limits an attacker to the area of that station's radio. Base stations are therefore incapable of obtaining authentication and ciphering information from, executing commands on, or persistently denying service to MM/GW components or users across an entire metropolitan area. The vulnerabilities we uncover in this paper grant an attacker several of these capabilities where the normal base station functionality could not.

Compromised access to a base station is a credible and increasingly common threat. Multiple incidents involving compromised cell tower equipment have surfaced in the last 5 years, and in 2021 the EU Agency for Cybersecurity (ENISA) reported that 18% of all telecommunications security incidents involved specifically targeted mobile base stations [21, 57]. Moreover, modern 5G base stations are deployed with much greater density and in more accessible and secluded areas than previous generations. Widespread femtocell offerings both in the US [10, 16, 52, 56] and worldwide [18, 25, 32, 42] additionally provide an attacker persistent physical access to such devices, which are susceptible to compromise via flash/RAM dumping and other exploits [17, 19, 26, 58].

2. Malicious UE. The NAS protocol conveys signalling information between the UE and the MM component, and is treated as opaque data by the intermediary base station. Thus, exploits that can be triggered via unexpected or malformed NAS payloads can enable an arbitrary UE to directly compromise a MM component. NAS datagrams are first exchanged between the MME and UE at the beginning of authentication procedures; as these packets must be parsed and interpreted by the MME before it can verify the legitimacy of the UE, we designate vulnerabilities in the NAS as remote pre-authentication attacks.

4 METHODOLOGY

Our methodology is as follows: we first run several LTE/5G testbeds to determine cellular-specific challenges to fuzzing (Section 4.1). From these observations, we design a netwok fuzzing harness to performantly fuzz cellular interfaces (Section 4.2), as well as a novel approach to structure-aware fuzzing of ASN.1-specified protocols (Section 4.3). Last of all, we procure fuzzing seeds both from manual testbed analysis and automated generation, as described in Section 4.4.

4.1 Challenges in Fuzzing LTE/5G Components

To inform our fuzzing approach, we first set up and configure several working LTE/5G cores as testbeds, complete with simulated base stations and radio connections to user devices (shown in Figure 2). Once all components in the cellular testbed are verified to be operational, we simulate various UE behaviors such as association, authentication, call and data transfer, and dropped connections. We capture and subsequently analyze message exchanges associated with these interactions. We then perform input stress testing specifically on RAN-Core interfaces to determine what bottlenecks would make fuzzing untenable. From these experiments, we discover several challenges to fuzzing cellular core interfaces:

Intentionally Delayed Messages. Some protocols, such as Stream Control Transmission Protocol (SCTP), define mechanisms that intentionally delay packet acknowledgement by hundreds of milliseconds to reduce total packet transmission. While such behavior

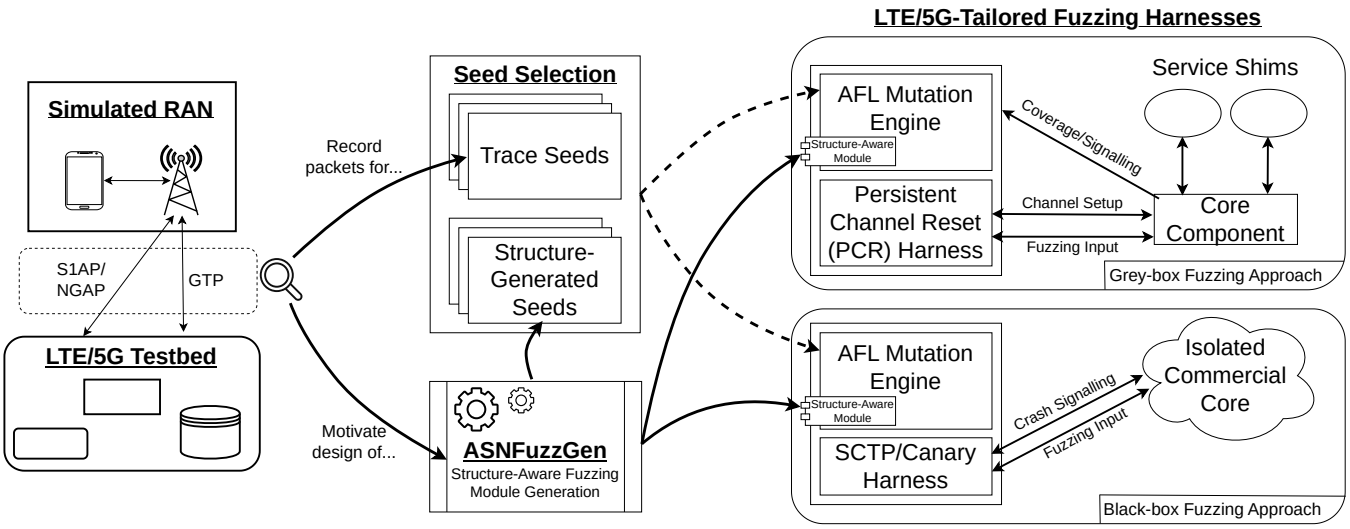


Figure 2: RANsacked System Overview. Based on observations of message flows between simulated RAN components and LTE testbeds, we approach RAN-core fuzzing by applying structure-aware fuzzing/seed generation and run campaigns using a specialized LTE-informed harness.

is desirable in the context of high-throughput cellular core operation, it severely degrades the speed at which network-oriented fuzzing can occur.

Complex Component Initialization and Architecture. In multiple cellular core implementations, certain components (such as the MM) are divided into several threads or even subprocesses with remote procedure calls defined between them. Such architectures introduce synchronization challenges during fuzzing engine initialization, reduce the efficacy of coverage-guided fuzzing (as coverage feedback is limited to one process), and introduce instability in coverage feedback due to multi-process concurrency. To add to this, many MM and GW component implementations introduce significant overhead during initialization each time they are started (hundreds to thousands of milliseconds) and consume a relatively large memory footprint, further stifling the potential for performant fuzzing using existing approaches.

Breadth and Depth of Protocol Structural Complexity. Many cellular protocols are the culmination of more than 30 years of additions to functionality in the cellular core. As a result of this, certain protocols have both a breadth of message types and a depth of nested field structure that proves difficult for standard fuzzing techniques to adequately explore. In particular, S1AP defines 91 distinct base message variants and over 1400 unique Information Element (IE)s of varying composition, many of which are reused dozens of times as fields for other IEs. NGAP similarly defines nearly a hundred base messages and over a thousand IEs, expanding the number of messages and IEs relative to S1AP. We observe that even after introducing more complex interactions into our testbed, only a small fraction of message types and IEs were captured in traces to be used as seeds for fuzzing.

Interface Channel Initialization. Both S1AP/NGAP and GTP require initialization messages to permit further communication on the interface. The S1AP requires an S1Setup message (or NGSetup for NGAP) that matches the configured identity of the MM component before processing further packets. Meanwhile, GTP cannot handle packets until a GTP tunnel is established, which is initialized by the MM component sending a PFCP control or GTP-C message to the GW component over a separate interface. As such, for a fuzzing harness to have any meaningful coverage depth, sufficient steps must be made by the harness to correctly initialize the interface it is targeting.

Interconnected Nature of Components. Both the MM and GW components have multiple defined interfaces with other internal core components. Several implementations manifest this dependency by refusing any inputs from RAN-Core interfaces until other core communication channels are established. On the other hand, implementations that permit communication without core dependencies end up omitting functions pertaining to those connections, leading to incomplete coverage.

4.2 Network Fuzzing Harness Design

Grey-box Fuzzing Harness. To overcome the challenge of passing input into the event-driven core components, we introduce a concurrent 'fuzz' thread to the component being fuzzed. This fuzz thread persistently iterates through inputs received from the AFL++ forkserver and establishes a distinct communication channel with the component under test for each input. We observe during testbed analysis that for both MM and GW interfaces, distinct connection channels isolate state from being left over by previous fuzzing rounds; leveraging this behavior enables us to efficiently fuzz while maintaining a reasonable degree of stability in coverage output.

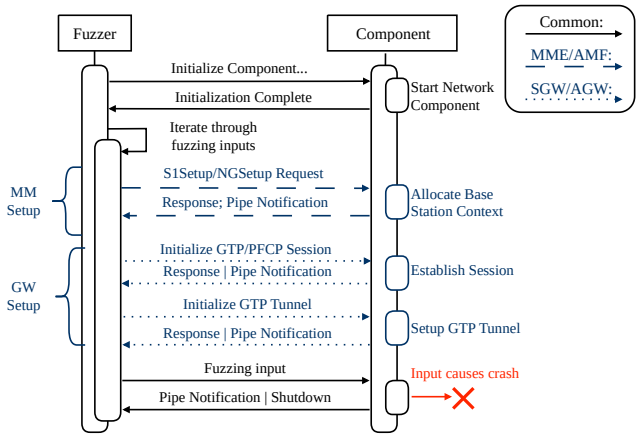


Figure 3: The combined workflows of MM and GW component Persistent Channel Reset (PCR) fuzzing harnesses. Common steps are illustrated in black, solid lines. Both methods follow an approach of establishing a communication channel for each fuzzing input and performing necessary setup prior to that input.

	AFLSmart [46]	Nautilus [7]	RedQueen [8]	Autogram [29]	Glade [11]	Grimoire [12]	ASNFuzzGen
Supports non-ASN.1 Formats	●	●	●	●	●	●	○
Works without Format Specification	○	○	●	●	●	●	○
Works with Binary-Only Targets	●	○	●	○	●	●	●
Works without Good Seed Corpus	○	●	●	○	○	●	●
Synthesizes Precise Grammar	●	●	○	●	●	●	●
Pluggable across Mutation Engines	○	○	○	●	●	○	●
Bit-precise Field Support	○	○	○	○	○	○	●

Table 1: Comparison of state-of-the-art grammar-aware fuzzing approaches. Filled circle indicates the property is supported, whereas empty circle indicates it is not.

To synchronize the beginning and end of each fuzzing input between the MM/GW components and the 'fuzz' thread, we introduce a pipe notification channel between the two. After a fuzz input has been received and the connection has been fully terminated, the MM/GW component sends a notification to the 'fuzz' thread indicating it will be idle and awaiting fuzzing input. The 'fuzz' thread awaits the reception of this notification before looping to its next fuzzing input. This ensures that no coverage information from a prior fuzzing input is erroneously recorded as being from subsequent fuzz input. The extent to which this approach enables effective cellular component fuzzing is evaluated in Section 5.5.

We hereafter refer to the combination of these techniques as Persistent mode with Channel Reset (PCR), as the communication channel is reset between each fuzzing input passed to the component. We apply our PCR harness to both MM and GW components. The complete control flow of PCR can be seen in Figure 3.

Blackbox Fuzzing Harness. As part of our research, we obtain remote access to fuzz proprietary commercial implementation of the MME. No coverage information can be obtained remotely from this core, so we use a modified version of AFLNet with disabled coverage to perform black-box fuzzing. This modified AFLNet runs 'havoc' routines randomly on seed inputs with no coverage guidance; as such, seed selection and structure-awareness is of particular importance in these experiments.

To detect crashes, we introduce a "canary" channel–an SCTP connection that exchanges S1Setup messages with the MME prior to each fuzzing input. After each fuzzing input channel is finished, the canary connection attempts to communicate with the MME. If successful, the fuzzer continues to test the next fuzzing input. In the event that the canary connection is closed or reset, the fuzzer marks the last tested input as a crash and waits for a timeout period before continuing to ensure that the crashed MME has restarted. This approach overcomes false negatives caused by race conditions with connecting to an MME after fuzzing (as the MME could have crashed and been reloaded). It also overcomes false positives wherein the MME abruptly closes fuzzing connections due to malformed input–since the canary is an unrelated channel, it should remain open even when a fuzzing channel closes.

4.3 Structure-Aware Fuzzing

As mentioned in Section 4.1, the S1AP and NGAP protocols include a significant breadth of IE field type and structure. These messages are encoded with ASN.1 PER, which compresses IE fields into bit-specific boundaries and adds precise padding to align certain IEs to word boundaries. Because of the strict correctness requirements enforced by PER, our preliminary fuzzing experiments showed that 96% of generated fuzzing inputs do not pass ASN.1 decoding, representing nearly two orders of magnitude of performance degradation in fuzzing post-decode logic. These challenges show up in many interfaces within cellular cores and RANs that use the same encoding, as in other fields such as space communications and automotive interfaces [1, 2, 22, 51].

We resolve this challenge by designing ASNFuzzGen, a code generation tool capable of parsing arbitrary ASN.1 specifications to emit a structure-aware module (as outlined in Figure 2). This module can be easily plugged into an existing fuzzing harness to enable the syntactically valid generation of ASN.1 packets, and is highly adaptable to any fuzzing engine that emits byte values as fuzzing inputs. ASNFuzzGen has been tested successfully on several other cellular protocols (E2AP, SUPL, NGAP and RANAP) and is applicable across diverse domains employing ASN.1 specifications.

We contrast ASNFuzzGen with other state-of-the-art approaches in Table 1. ASNFuzzGen notably can generate the bit-precise structures necessary for ASN.1 PER, whereas current approaches assume structures that are aligned to byte boundaries. This limitation precludes the use of these approaches to fuzz PER constructs. For instance, length fields for variable-length IEs are packed in the minimum number of bits needed to unambiguously represent them; as

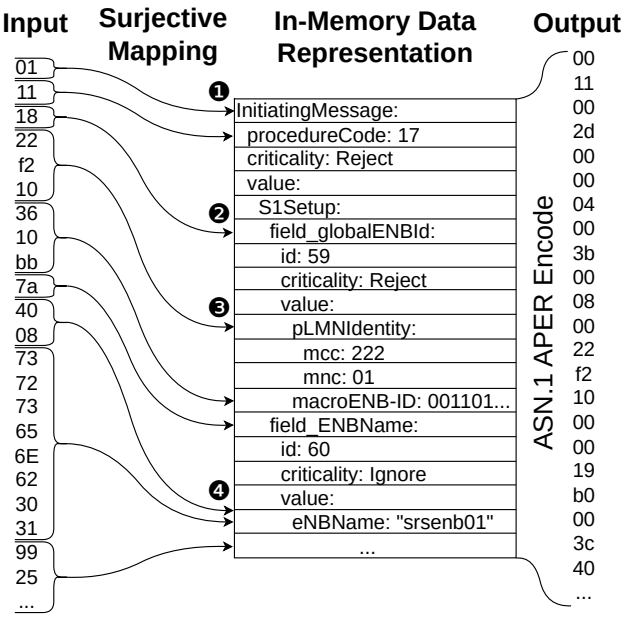


Figure 4: The ASNFuzzGen module maps raw byte inputs from AFL to semantic decisions on the existence, length and content of data fields. The resulting output is guaranteed to be well-formed ASN.1, thereby enabling much deeper state to be reached early on in fuzzing.

such, they cannot be modeled by a byte-precise grammar. ASNFuzzGen additionally integrates seamlessly within existing fuzzing mutators (whereas other approaches normally necessitate custom mutation engines). While other approaches are more suitable across many general byte-precise grammars and protocols, ASNFuzzGen offers the first bit-precise structure-aware approach for fuzzing arbitrary ASN.1, which makes it well-suited for a wide array of cellular, space, vehicular and other protocols.

Functional Mapping of Inputs to ASN.1 Outputs. Each module emitted by ASNFuzzGen contains a function f that defines a surjective mapping from an arbitrary-length sequence of bytes into an intermediary in-memory data structure conforming to ASN.1. This data structure is then encoded using ASN.1 PER to output a sequence of structured bytes. As shown in Figure 4, input bytes are consumed sequentially in order to: (1) choose a specific enumerated value out of several options, (2) determine whether an optional field should be present or not, (3) populate fields with data that conforms to composition requirements, or (4) choose a length for variable-length fields.

In addition to adhering to the ASN.1 specification for data constraints, we include a postprocessing step in ASNFuzzGen for S1AP/NGAP that accounts for special rules in packets not adequately enforced by the ASN.1 definition. In particular, certain fields are marked as 'Mandatory', but no guarantee is enforced that such fields will be present (we explore the implications of this on code security in Section 6.6). ASNFuzzGen accounts for this by parsing 'Mandatory' labels from IE fields and adjusting the forward/inverse mapping functions it emits to include all mandatory fields by default, adding a 1/32 chance of omitting the field based on a consumed input byte. By including this step, we enable fuzzing to explore conditions beyond mandatory field checks while still catching bugs introduced by the absence of those fields.

As the NAS protocol has no formal ASN.1 specification, ASNFuzzGen does not apply structure-aware procedures to it. Rather, NAS payloads are treated as a variable-length field of raw bytes by ASNFuzzGen, thereby enabling unstructured fuzzing of NAS inputs within syntactically-valid S1AP/NGAP payloads.

Partial Inversion Function for Mapping Seeds. Regular seeds cannot be used in a structure-aware fuzzer, as they will be mapped into new bytes instead of being passed transparently to the MM component. To overcome this, we also include a partial inverse function in the module, g, such that $f(g(input)) = input$ (though not necessarily $g(f(input)) = input$). One can simply pass selected seeds through g to obtain "inverted" seeds prior to fuzzing. Using these inverted seeds for structure-aware fuzzing will result in the original seed values being passed onward to the system under test.

4.4 Seed Selection

We gather S1AP/NGAP and GTP packets exchanged between a simulated RAN and active LTE/5G testbed to use as one selection of seed inputs for fuzzing. Taken together, these messages comprise necessary interactions a base station would make with the core through the connection lifecycle of a mobile device. We likewise use these same seeds in our black-box S1AP fuzzing experiments.

In addition to this "Trace" set of seeds, we employ our structure-aware module to generate a wide variety of syntactically-valid S1AP/NGAP control messages from thousands of random inputs. We then trim this set down to two control messages of each base type and use these as a second "Generated" set of starting seeds. For structure-aware fuzzing campaigns, we use the S1AP or NGAP module's partial inverse function to invert each of each of these seed sets, thereby ensuring the same resulting packet sequence. We hypothesize that this set will improve the discovery of vulnerabilities stemming from uncommon packet types.

In adhering to best practices in fuzzing research [36], we run a third set of fuzzing campaigns for each core using an empty seed corpus. This serves as a benchmark to evaluate our other seed selection approaches against.

5 EVALUATION

In evaluating the efficacy of our fuzzing methodology, we consider the following research questions:

- **RQ1** Are LTE/5G implementations robust against inputs originating from the RAN?
- **RQ2** Can we discover additional vulnerabilities and coverage paths by applying structure-aware approaches to fuzzing?
- **RQ3** Does our fuzzing harness efficiently fuzz RAN-core interfaces?
- **RQ4** Does grey-box fuzzing of open-source cellular implementations inform more effective black-box fuzzing?

Table 2: Cellular components/interfaces tested by RANsacked (marked with ✓). Magma and OAI share the same MME, so subsequent discussion will combine the two.

Name	Version	Fuzz Approach	S1AP	NGAP	GTP
Magma	1.8.0	Coverage-Guided	✓	✓	-
NextEPC	1.0.1	Coverage-Guided	✓	N/A	✓
Open5GS	2.6.1	Coverage-Guided	✓	✓	✓
OAI	2.0.0	Coverage-Guided	✓	✓	✓
SD-Core	1.3.0	Coverage-Guided	✓	-	-
srsEPC	20.10	Coverage-Guided	✓	N/A	✓
Athonet vEPC	9.4.0	Remote Black-box	✓	N/A	-

We answer **RQ1** in Section 5.2, and further explore its ramifications in Section 6. **RQ2** is addressed in Section 5.3 and 5.4, while **RQ3** is covered in Section 5.5 and **RQ4** in Section 5.6.

5.1 Experimental Setup

We apply our grey-box fuzzing harness to all six publicly-available LTE core implementations (Magma, NextEPC, Open5GS, Open Air Interface (OAI), SD-Core, and srsEPC) and three 5G implementations (Magma, Open5GS, and OAI), and perform black-box fuzzing on a proprietary remote core for which no source or binary access is given (HPE Athonet vEPC). Of these implementations, we note that at least four of these (Open5GS, Magma, OAI, Athonet) are used in commercial/industrial applications and are thus representative of "in-the-wild" systems rather than merely hobby or research projects [9, 13, 37, 43, 50]. We fuzz the MM component of every core (with Magma and OAI sharing an MME implementation), as well as the GW component where possible. Table 2 lists the versions of software fuzzed and indicates covered interfaces.

We run a total of 512 48-hour grey-box fuzzing campaigns spread across six open-source LTE core and three 5G core implementations. For each MM component, we run 30 distinct regular fuzzing campaigns, i.e. 10 for each seed selection (Null, Trace and Generated). We then repeat this process with the ASNFuzzGen structure-aware module added to the fuzzer. For GW components, we run eight distinct fuzzing campaigns (four with seeds, four without). Each campaign is run using AFLPlusPlus version 4.04c, with a persistent-mode loop set to 1000 iterations and minimum/maximum input sizes of 1 and 100k bytes, respectively. We run our experiments on a Supermicro H12SSW-NT rack-mounted server running Ubuntu 20.04 with an AMD EPYC 7763 64-Core CPU, 256GB of DDR4 3200 ECC RAM, and a Samsung 870 EVO Solid-State Drive. Grey-box fuzzing campaigns are isolated in separate Docker containers limited to one distinct CPU core each to ensure uniformity of test conditions. Black-box campaigns are performed on a standalone Athonet vEPC core on an isolated network.

5.2 MM/GW Implementation Robustness

Our fuzzing campaigns uncover 119 vulnerabilities across seven cellular core implementations, nearly all of which are previously undiscovered. 4 vulnerabilities stemmed from GTP parsing within SGW components (2 each for Open5GS and OAI). The remaining 115 vulnerabilities are spread across seven MME and three AGW implementations; a detailed listing of these vulnerabilities can be viewed in the extended version of this paper. With the exceptions of srsEPC and OAI 5G, *nearly every MME contained vulnerabilities at the NAS layer that could be exploited by an unauthenticated UE.* We explore and analyze these further in Section 6. Our findings indicate that many cellular cores lack robustness against malicious RAN-core inputs (**RQ1**).

To better explore the impact of our fuzzing approaches on vulnerability discovery, we perform a wide-scale analysis of approximately 37,000 crashes saved by AFL across the 300 MME/AMF grey-box fuzzing campaigns. To map crashes to distinct vulnerabilities, we rerun saved crashes on a patched version of the tested component that emits the ID of the reached vulnerability in logs. Following this, we extract the timestamp and execution count of the earliest recorded crash for each vulnerability across all campaigns.

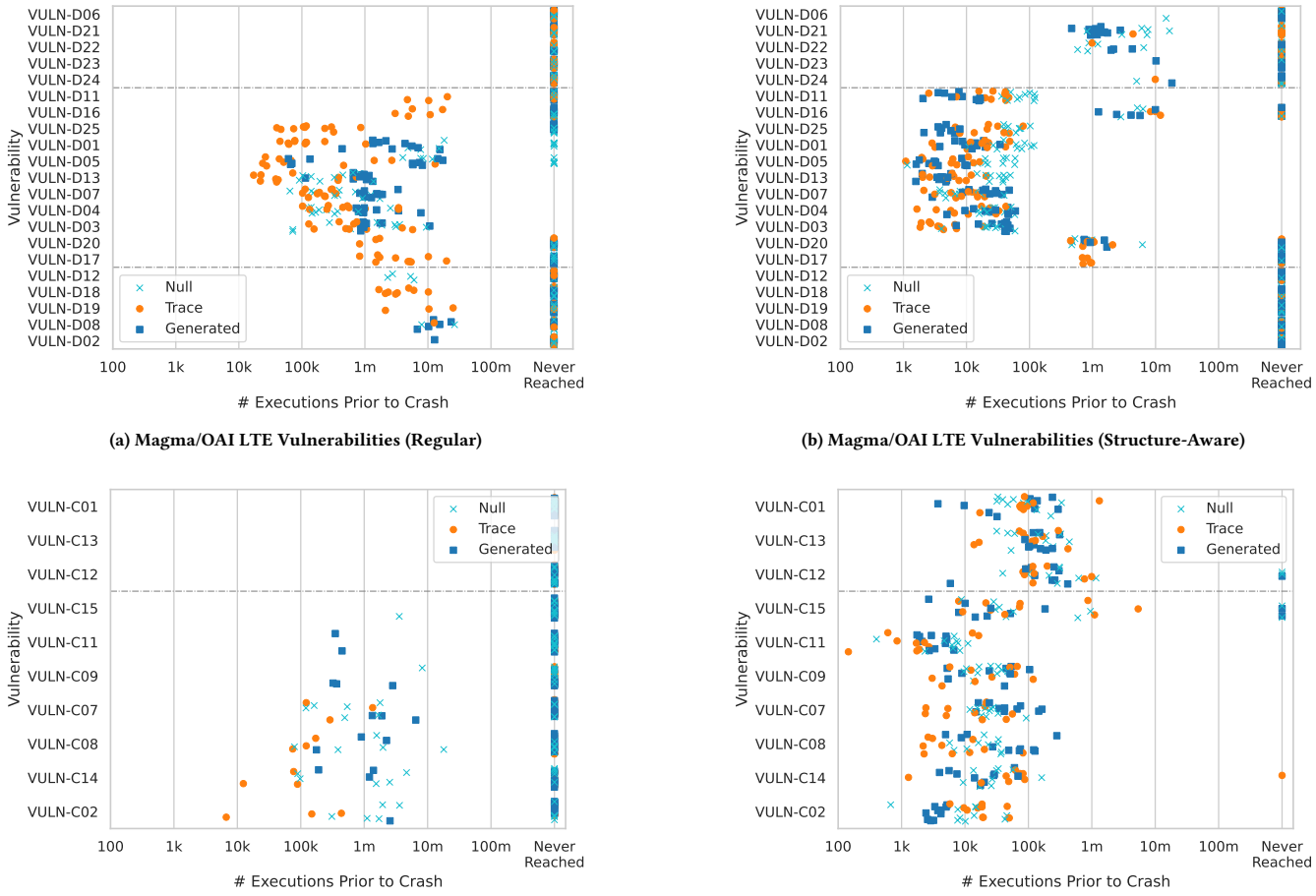
Figure 5 shows the result of this analysis for Open5GS and Magma; we include findings from NextEPC and SD-Core in the extended version of this paper. We omit srsEPC, as only one vulnerability was found across all fuzzing campaigns for it. We observe that several vulnerabilities are uncovered during fuzzing only under specific conditions and configurations. We explore the discovery of these vulnerabilities in the context of specific seed selections in Section 5.4, the introduction (or absence) of structure-aware fuzzing in Section 5.3, and the number of executions the campaign can carry out in Section 5.5.

5.3 Structure-Aware Fuzzing

Effect on Coverage. For all MM implementations, structure-aware fuzzing outperformed regular fuzzing significantly during early hours and by a modest margin through the remainder of fuzzing campaigns. Figure 7 shows coverage over time for Magma LTE; coverage across other campaigns follow largely the same trend, and can be viewed in the extended version of this paper. For each seed approach, the minimum and maximum observed coverage out of the 10 runs are indicated by a shaded area; the line within that shaded area represents the average coverage across 10 runs. This increase in coverage comes *in spite of* the fact that the structure-aware fuzzer does not cover basic blocks reachable by invalid ASN.1 inputs; these basic blocks number in the several hundreds.

The low percentile coverage of basic blocks observed across implementations is a direct result of the span of functionality the MM and GW components cover. RAN-facing interfaces and handlers are but one of nearly a dozen different interfaces used to communicate with various core components. Many of these interfaces require large libraries of auto-generated code to handle distinct protocols such as Diameter, GTP-C and HTTP REST interfaces; combined with the actual implementation of interface functionality, these inflate the total number of basic blocks any given MM or GW implementation has. While this makes it difficult to quantify the extent to which our fuzzing completely covers RAN-facing functions, subsequent analysis in Section 6.3 indicates our fuzzing uncovered vulnerabilities spanning dozens of message types and extending deep into protocol handlers.

Detected Crashes. Following the same trend as coverage, distinct crashes are found much earlier in fuzzing when the ASNFuzzGen module is applied to each of the cores: the final number of saved



(a) Magma/OAI LTE Vulnerabilities (Regular)

(b) Magma/OAI LTE Vulnerabilities (Structure-Aware)

(c) Magma 5G Vulnerabilities (Regular)

(d) Magma 5G Vulnerabilities (Structure-Aware)

Figure 5: Vulnerabilities discovered across Magma LTE and 5G fuzzing campaigns. Grey dotted lines separate vulnerabilities by those found only with structure-aware fuzzing (top), those found with both approaches (middle), and those found only with unstructured fuzzing (bottom).

crashes ranging anywhere from a modest 15% increase in Open5GS to a 300% increase in Magma for trace seeds. These crashes correspond to new distinct vulnerabilities discovered in each core. As an exception, we note some vulnerabilities that structure-aware fuzzing could not detect–in particular, Open5GS, SD-Core and srsRAN all contained vulnerabilities in the ASN.1 decoding function for S1AP that required malformed inputs to reach. ASNFuzzGen produces only syntactically-valid ASN.1, and therefore would not produce the invalid ASN.1 necessary to reach these vulnerabilities.

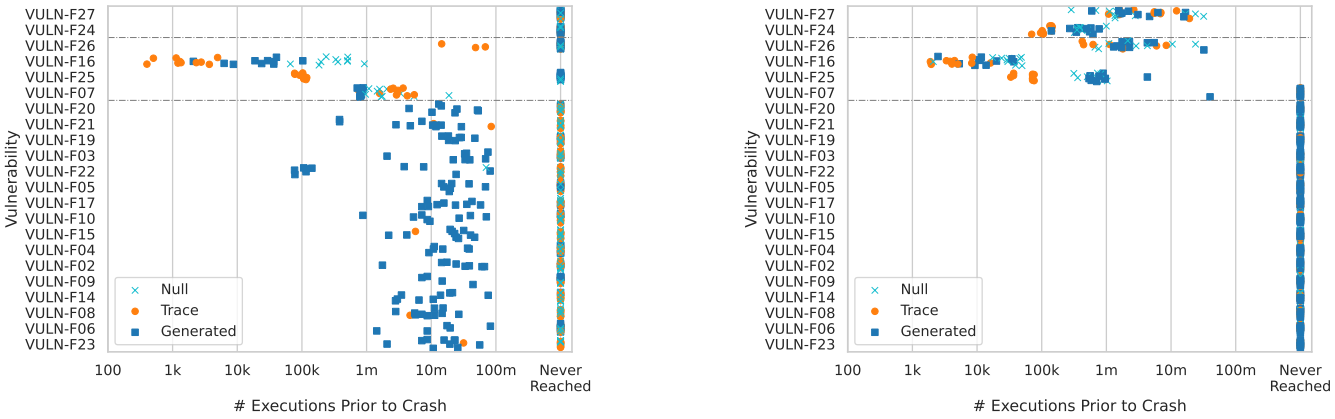
Takeaway. Fuzzing campaigns that make use of the ASNFuzzGen module exclusively discover 17 new vulnerabilities, with nine of these being high-impact memory corruption vulnerabilities in NAS payloads (**RQ2**). Vulnerabilities found by regular fuzzing but not by structure-aware fuzzing were related either to ASN.1 decoding errors or missing Mandatory IE fields, which both occur relatively early on in packet processing.

5.4 Seed Selection

Impact on Coverage. As Figure 7 shows, initial seed corpus makes a significant impact on coverage for fuzzing campaigns without the ASNFuzzGen module. Across MM components tested, trace-derived seeds strongly outperformed structure-generated seeds in all but SD-Core. However, we observe in Figure 5 that this does not always correlate to the discovery of more vulnerabilities.

The choice of seed corpus had a far less pronounced impact on code coverage when the ASNFuzzGen module was applied. In all MM implementations other than Magma/OAI, the choice of seed corpus made no significant difference in coverage after the first few hours of fuzzing. Magma/OAI shows some coverage improvement in generated seeds over seeds recorded from packet traces, though the effect is muted compared to fuzzing without the ASNFuzzGen module.

Since the GW component handles packet in a less complex fashion, we generate seeds for GTP fuzzing by enumerating all GTP



(a) Open5GS LTE Vulnerabilities (Regular)

(b) Open5GS LTE Vulnerabilities (Structure-Aware)

Figure 6: Vulnerabilities discovered in Open5GS LTE fuzzing campaigns. Vulnerabilities are ordered vertically based on which approach (structured, both or unstructured) discovered the vulnerability.

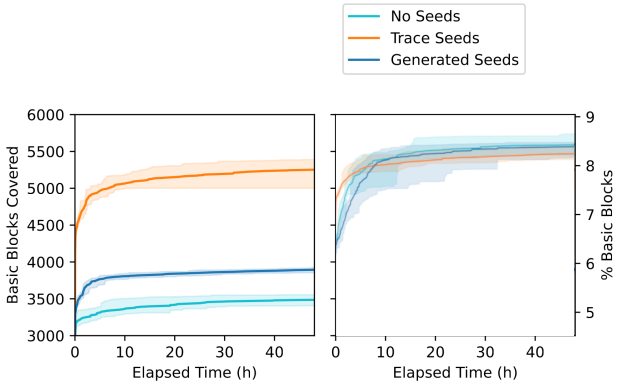


Figure 7: Magma Basic Block Coverage Over Time. Colored areas represent the min/max spread of block coverage across 10 distinct fuzzing runs, while solid lines represent mean block coverage. Left graph shows the result of regular fuzzing, while right graph shows structure-aware results.

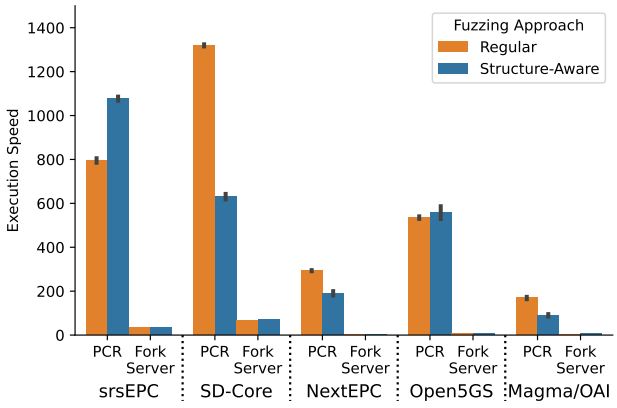


Figure 8: Fuzzing harness performance comparison between Persistent Channel Reset (PCR) and traditional fork server harnesses across five open-source MME implementations.

headers with attached random data. We find that fuzzing with generated seeds improves the block coverage marginally and shortens execution time, but most gains in coverage occur very early on across all GTP fuzzing campaigns regardless of seed selection.

Takeaway. We observe a significant number of vulnerabilities in Open5GS LTE (shown in Figure 6a) that are only discovered via the use of structure-generated seeds in regular fuzzing. These vulnerabilities are common in their preconditions: each requires a valid packet with a particular missing 'Mandatory' field and an invalid field value. We note the presence of vulnerabilities in other LTE and 5G cores found only with generated seeds, and even vulnerabilities in structure-aware fuzzing that required generated seeds to reach. Thus, we observe that our generative approach to improve the breadth of seed selection proves effective at discovering new vulnerabilities (**RQ2**). At the same time, we note that the effect of seed selection is somewhat more muted in structure-aware fuzzing rounds. Combined with observations on structure-aware fuzzing improvements, we conclude that using a combination of regular fuzzing with generated seeds and structure-aware fuzzing may lead to the widest breadth of discovered vulnerabilities.

5.5 Harness Performance

We benchmark the performance of RANsacked using our PCR persistent fuzzing harness approach vs. traditional fork server/stdin approaches (e.g. those used by AFLNet). As seen in Figure 8, PCR results in up to a 100x improvement in performance over fork server fuzzing, with hundreds to thousands of inputs tested each second. We set design persistent fuzzing to execute 1000 iterations

per fork in this benchmark. Better performance can potentially be attained by increasing the iteration count, at the cost of possibly lower coverage feedback stability.

The synthesis of this benchmark and Figure 6 paints a clear picture of the vulnerability discovery made possible by the more performant PCR fuzzing approach. NextEPC, Magma/OAI and Open5GS benchmark at 1.9, 1.1, and 5.8 executions per second respectively without PCR; over the course of a 48-hour fuzzing campaign, this translates to 190k-1.0m total executions. Between these three implementations, less than half of all vulnerabilities would have been discovered within the 48 hour window, and a significant number of repeat findings across distinct fuzzing campaigns would likewise be absent. In some cases, such as VULN-D23 or VULN-F05, fuzzing campaigns would have had to run for upwards of *a hundred days* to discover the same vulnerabilities found by PCR in under 48 hours. Overall, we see that our PCR fuzzing harness achieves significantly faster execution speeds, which enables the discovery of vulnerabilities that would not be otherwise found (**RQ3**).

Effect of PCR on fuzzing stability. One concern with persistent-mode fuzzing is the introduction of latent state in earlier fuzzing executions that lead to differing coverage output for the same fuzzing input. The extent to which a program always produces the same coverage feedback for a given input is termed as "stability" in fuzzing engines such as AFL++, and it is measured by the percentage of branches that always give consistent feedback for an input. Maintaining a sufficiently high level of stability is important, as fuzzing engines must either a) ignore unstable branches and risk missing inputs that effectively explore new coverage in those branches, or b) allow unstable branches and risk scheduling many inputs that trigger unstable branches without meaningfully exploring new program states.

Several of our MM component fuzzing campaigns run at greater than 80% stability. For those that exhibited lower stability, we discovered that the the MM was architected as several components that would concurrently execute and communicate via remote procedure calls, which led to inherent coverage instability independent of the application of PCR. Executing multi-threaded and multi-process fuzzing in a way that ensures stable and deterministic coverage output is an open challenge across many fuzzing works and tools. While RANSacked is able to maintain high stability in single-threaded environments, we consider multi-threaded stability to be outside the scope of our work.

5.6 Black-box Fuzzing

Using a subset of the approaches demonstrated to yield increased coverage for grey-box fuzzing, we employ remote black-box fuzzing against an instance of the HPE Athonet core. We separate these strategies garnered from grey-box fuzzing into three distinct approaches: structure-aware fuzzing using the ASFuzzGenn generated module, regular fuzzing using trace seeds from testbed observations, and regular fuzzing using randomly generated S1AP packets of each packet type as seeds.

For each approach, we run fuzzing campaigns with a timeout of 20,000 iterations (roughly 1.5 hours of fuzzing) over 20 runs. Using these approaches, we identify a total of eight unique crashing inputs. The baseline method of fuzzing yielded three unique crashes, while adding structure-aware fuzzing to AFLNet uncovered an additional two unique crashes. The introduction of generative syntactically-valid seeds produced the strongest results in black-box fuzzing, resulting in the discovery of an additional 3 unique vulnerabilities. To add to this, one of the vulnerabilities discovered through generative seed selection pertained to the NAS portion of the S1AP packet, which is passed transparently from UE through the base station to the MME.

These findings are consistent with our general hypothesis that a structure-aware approach in both seed selection and fuzzing generation would be instrumental in finding S1AP and NAS vulnerabilities (**RQ4**). We note that the three vulnerabilities discovered without the assistance of ASNFuzzGen all pertain to ASN.1 Information Element ID fields in message types that were directly observed in the testbed. Meanwhile, fuzzing with structure-generated seeds yielded vulnerabilities in more diverse message types in addition to rediscovering the crashes found by trace seeds. The addition of the structure-aware fuzzing module in black-box fuzzing netted vulnerabilities beyond ASN.1 decoding, such as crashes related to attempting to release an unassociated UE or transferring configuration parameters to a nonexistent base station.

6 VULNERABILITY ANALYSIS

We analyze the crashes discovered as a direct result of our fuzzing campaigns and perform root cause analysis of each crash to remove any duplicates. For open-source cores, we pinpoint the exact location in code that causes the MM or GW component to crash and categorize these by vulnerability type, cause, and general location in the network component. For the proprietary MME implementation, we narrow the vulnerability down to the specific field or value change that causes the crash by repeatedly testing small modifications to the packet on the black-box. We then perform a best-effort determination of the likely cause and location of the vulnerability based on the crashing field/value.

Of the 119 vulnerabilities RANSacked discovered, 79 were found in MME implementations, 36 in AMF implementations and four in SGW implementations. 25 vulnerabilities lead to NAS pre-auth attacks that can be carried out by an arbitrary cellphone. Nearly all of these vulnerabilities were as of yet undiscovered at the time we disclosed them to the respective maintainers of these cellular cores.

6.1 Vulnerability Classes

For each vulnerability, we identify the root cause of programming errors that lead to the manifestation of the vulnerability and classify them. Table 3 categorizes these vulnerability types for each implementation fuzzed. We find that 37 of the discovered vulnerabilities trigger invalid memory use or memory corruption, which could potentially be used as gadgets to facilitate remote code execution. We experimentally demonstrate such an attack in Section 6.5. All discovered GW vulnerabilities are reachable assertions triggered as the result of unexpected protocol state, while a majority of memory corruption issues occurred within NAS handling methods. NAS vulnerabilities are all reachable by messages sent prior to authentication between the UE and MM component.

Table 3: Classes of Vulnerabilities Discovered in MM

Implementation	Reached Assertion	Null Dereference	Type Confusion	Uninit. Pointer	OOB Read	OOB Write
Magma/OAI LTE	2	13	1	0	4	2
Open5GS LTE	26	0	1	0	0	0
NextEPC	0	0	7	0	0	2
SD-Core	0	0	1	0	0	8
srsEPC	0	0	0	0	0	1
Magma 5G	19	0	0	0	0	1
OAI 5G	0	2	0	5	1	2
Open5GS 5G	5	0	0	0	0	0
Total	52	15	10	5	5	16

Table 4: Cause of MM Vulnerabilities

Implementation	Invalid Codegen	Absent Field	Malformed Field	Malformed Size	Unexpected State	Other
Magma/OAI LTE	0	12	6	3	2	0
Open5GS LTE	1	21	2	1	0	0
NextEPC	7	0	2	0	0	0
SD-Core	1	0	5	2	0	1
srsEPC	1	0	0	0	0	0
Magma 5G	0	19	0	1	0	0
OAI 5G	0	4	2	1	4	0
Open5GS 5G	0	0	4	1	0	0
Total	10	56	21	9	6	1

With regards to causes, Table 4 categorizes common root causes of vulnerabilities across fuzzed cores. Several Type Confusion vulnerabilities stemmed from faulty ASN.1 code generation, leading to a mismatch in type for some IEs when interpreting and subsequently freeing memory. Several implementations lacked any bounds checks when copying variable-length S1AP/NAS IEs into fixed-length buffers (OAI, SD-Core, NextEPC) or else contained integer underflow bugs that rendered such checks useless (Magma). Most reachable assertions and null pointer crashes were due to expected missing 'Mandatory' IE fields; we discuss this further in Section 6.6. Taken as a whole, these vulnerabilities outline a pattern of implicit trust in RAN inputs by MM implementations.

Since GW components mainly perform as a packet forwarding module without complex decoding functionality, they expose less attack surface compared to S1AP/NGAP. Our evaluation finds four distinct crashes in Open5GS and OAI, all caused by reachable assertions in GTP packet handlers. Crashes in Open5GS occur when the GTP packets apply for unavailable resources, such as when a GTP packet attempts to carry user-plane data from UE to the core network without an established GTP tunnel. For OAI, all crashes are caused by the misuse of open-source libraries, including wrong inputs in *spdlog* [40] and bad tunnel queries in the key-value store – TEID (GTP tunnel ID) and PDR (packet detection rule) – of *folly* [23]. The root causes of these are unexpected states.

6.2 Breadth of Vulnerable Message Types

As previously discussed, both S1AP and NGAP each define nearly a hundred base message variants each and well over a thousand IE data types. The vulnerabilities we discover span 32 of these base message variants. Of these, most discovered vulnerabilities occurred while handling InitialUEMessage payloads (responsible for conveying initial NAS payloads from the UE) which contained 22 distinct vulnerabilities scattered across various decoding and handling subroutines in implementations.

We similarly find vulnerabilities that are triggered by malformed or missing data across 29 distinct IE data types. Several IEs within the NAS are commonly mishandled by two or more implementations, such as the 'Emergency Number List' and 'IMSI' types lacking length bound checks, and generally oversized NAS payloads cause 4 of the found vulnerabilities.

Overall, the presence of vulnerabilities across a wide range of both message types and IEs points to the need for a fuzzing approach that enables broader and more comprehensive S1AP/NGAP fuzzing, such as what ASNFuzzGen enables. Many message handling routines both require complex input structures and modify global state during the course of handling. As such, alternative function-specific fuzzing strategies would be both tedious to implement and difficult to scale given the wide breadth of handler routines to cover. As such, the wide spread of vulnerabilities across many message types demonstrates the advantage that generalized protocol-targeted fuzzing RANsacked provides over function-specific fuzzing strategies.

6.3 Vulnerability Location/Depth

We identify four primary locations in which most MM component implementations contained vulnerabilities: in the core mechanics of the network daemon, in ASN.1 deserialization methods, in handlers for various S1AP/NGAP request types, and in NAS deserialization and handling. Table 5 breaks down vulnerabilities from each MM component by their location. These locations represent distinct layers of 'depth' of packet processing. In the leftmost column, 'Daemon' level vulnerabilities could be reached prior to full decoding by ASN.1 methods and therefore were relatively low complexity. On the other hand, 'NAS Handler' vulnerabilities required structurally correct ASN.1, present S1AP/NGAP mandatory IEs, correct IE field information, and partially correct NAS fields leading up to the source of crash. Thus, from left to right, each location in Table 5 is progressively 'deeper', or further on in packet processing, than the last.

We note a few trends in vulnerability location based on this categorization. First, nearly every implementation contained at least one bug in ASN.1 decoding that could be leveraged to trigger memory corruption. These relatively 'shallow' bugs were contained within code structured in such a way that it could be easily fuzzed using function-level fuzzers. Second, a strong majority (82%) of vulnerabilities were post-ASN.1 and 21% of S1AP/NGAP vulnerabilities occurred in NAS handling routines. This confirms our hypothesis

Table 5: MM Vulnerabilities by Location

Implementation	Daemon*	ASN.1 Decoder	S1AP Handler	NAS Handler
Magma/OAI LTE	0	0	13	12
Open5GS LTE	1	1	21	4
NextEPC	0	7	0	2
SD-Core	1	1	5	2
srsEPC	0	1	0	0
Athonet	0	5	2	1
Magma 5G	0	0	19	1
OAI 5G	3	1	7	0
Open5GS 5G	0	0	2	3
Total	5	16	69	25

* Vulnerabilities in the network event handler or memory pool

that fewer vulnerabilities would be due to ASN.1-specific errors and that enforcing correct ASN.1 structure through ASNFuzzGen would therefore improve crash discovery.

6.4 Case Study I: Type Confusion in Magma

LTE NAS messages can be one of two variants: EPS Mobility Management (EMM) or EPS Session Management (ESM). The message variant is normally encoded in a discriminant value in the header prepended to the message, such that the MME can interpret the message contents correctly. When encrypted, a NAS message has two of these headers: one preceding the encrypted payload and one encapsulated within the payload.

In Magma MME, RANsacked uncovered a vulnerability in the way encrypted NAS messages are decoded. Specifically, a header discriminator value contained within the encrypted payload is used to decode the data into memory fields, but a separate discriminator contained in the outer (unencrypted) header is used to subsequently interpret the decoded message struct. As a result, a payload with an outer EMM header and inner ESM header can cause pointer and data type confusion in the fields of that packet. An attacker could exploit this to read/write or `free()` memory at arbitrarily chosen pointer locations with a high degree of flexibility.

We note such a vulnerability is only possible due to the distinct structure of NAS–payloads need to be either plaintext or ciphered depending on the state of the connection with the UE, and messages can be one of two variants (EMM or ESM). The presence of multiple encapsulated headers with potentially different protocol discriminant values directly led to incorrect handling of such messages as seen in this case study. The 3GPP specification on NAS for LTE reads: "The protocol discriminator in the header of a security protected NAS message is encoded as "EPS mobility management messages." [3]. For ESM messages that are ciphered, this results in a protocol mismatch between encrypted and unencrypted variants of the same NAS payload. Clearer disambiguation may be needed in the specification with regards to handling outer vs. inner protocol headers.

```
482  case ProtocolIE_ID_id_NAS_PDU:
483  {
484      NAS_PDU_t *s1apNASPDU_p = NULL;
485      if(UplinkNASTransport_IEs__value_PR_NAS_PDU == ie_p->value
             .present)
486      {
487          s1apNASPDU_p = &ie_p->value.choice.NAS_PDU;
488      }
489      else
490      {
491          log_msg (LOG_ERROR, "Decoding of IE NAS PDU failed");
492          return -1;
493      }
494
495      proto_ies->data[i].IE_type = S1AP_IE_NAS_PDU;
496      memcpy(s1Msg->nasMsg.nasMsgBuf, (char*)s1apNASPDU_p->buf,
               s1apNASPDU_p->size);
497      //      ^ nasMsgBuf is fixed-size buffer; oversized NAS PDU
                leads to buffer overflow
```

Listing 1: Buffer overflow in SD-Core *Uplink NAS Transport* handler (line 496, *memcpy*).

6.5 Case Study II: Stack Overflow in SD-Core

When handling *Uplink NAS Transport* messages, handlers in SD-Core copy the contents of the NAS payload into a fixed-size buffer of 500 bytes (shown in Listing 1). No length check is performed prior to this copy, so excess bytes are copied from the NAS packet into stack memory. This overflow occurs within a loop that handles all IEs within the S1AP message. Following the buffer overflow, certain data pointers may be arbitrarily overwritten such that a subsequent *memcpy()* call actually overwrites the Global Offset Table; from there, a third *memcpy()* call is overwritten to call *system()*, leading to arbitrary code execution. We release a proof-of-concept implementation demonstrating this exploit as part of our vulnerability disclosure, but limit the exploit to require that certain stack protection features be disabled at build time in order to hamper any malicious use on production systems.

Several MM implementations exhibited unrestricted buffer overflow vulnerabilities similar to this case study, both in S1AP/NGAP and NAS-specific fields. The effect of such vulnerabilities being compromised is potentially catastrophic; any unauthenticated cellphone could send a payload triggering these memory corruption vulnerabilities, and the right input could give an attacker a foothold in the MM component, as we demonstrate with SD-Core. An attacker with remote access to an MM component could perform widespread surveillance across a metropolitan area, selectively deny service or manipulate authentication/billing against select targets, or pivot to attacks against the core's subscriber database (which services areas spanning an entire nation).

6.6 Common Vulnerabilities Across Cores

Through our analysis of vulnerabilities in open-source and commercial MMEs, we identify three domain-specific root causes of vulnerabilities that span multiple implementations.

1. Underscrutinized NAS Input. Four MM implementations contained unbounded buffer overflows across various fields in NAS payloads. An additional two implementations contained crashing assertions when certain unimplemented fields in the NAS were present. Given that the NAS allows untrusted, unauthenticated inputs to be passed into the core, it is telling that low-hanging vulnerabilities still show up in some of the most dangerous code of

```
S1AP-PROTOCOL-IES ::= CLASS {
  &id           ProtocolIE-ID UNIQUE, &criticality Criticality,
  &Value,                             &presence    Presence
} WITH SYNTAX {
  ID          &id
  CRITICALITY &criticality
  TYPE        &Value
  PRESENCE    &presence --- denotes Required/Optional IE fields
}

ProtocolIE-Container {S0AP-PROTOCOL-IES : IEsSetParam} ::=
  SEQUENCE (SIZE (-1..maxProtocolIEs)) OF
  ProtocolIE-Field {{IEsSetParam}}

--- &presence omitted when defining concrete types (seen below)
ProtocolIE-Field {S0AP-PROTOCOL-IES : IEsSetParam} ::= SEQUENCE {
  id S0AP-PROTOCOL-IES.&id (
    {IEsSetParam}
  ),
  criticality S0AP-PROTOCOL-IES.&criticality (
    {IEsSetParam}{@id}
  ),
  value S0AP-PROTOCOL-IES.&Value (
    {IEsSetParam}{@id}
  )
}
```

Listing 2: S1AP ASN.1 specification outlining mandatory field markers. As seen above, fields denoted as mandatory are not necessarily verified to be present by generated ASN.1 code.

nearly every LTE/5G implementation. Had these vulnerabilities existed in common webserver software, they would likely have been discovered and exploited 20 years ago; the breadth of vulnerabilities found in MME implementations highlights a significant need for more rigorous security analysis in cellular components and further application of security hardening techniques to the domain.

2. Unclear Invariants in S1AP/NGAP Specifications. The 3GPP specification for S1AP/NGAP indicates whether a given IE field is mandatory, optional or conditional using a distinct 'presence' field contained in the PROTOCOL-IES class. However, the underlying implementations of this class that define IE types during ASN.1 compilation omit the field, leading to no 'presence' information in implemented types (see Listing 2). The specification additionally reads as follows: "IEs marked as Mandatory (M) shall always be included in the message." A developer not familiar with the finer details of ASN.1 compilation may easily mistake this to mean that Mandatory IEs are enforced similarly to that of other ASN.1 constructs. Two MME implementations and one AMF contained multiple vulnerabilities stemming from the assumption of Mandatory field presence, indicating that insufficient clarity in this specification has a carryover effect on the robustness of implementations used in the wild.

3. Insecure/Invalid ASN.1 Deserialization. Despite years of work pointing to ASN.1 parsing as a key source of memory corruption vulnerabilities in cellular systems [55], such issues continue to plague cellular cores in use today. Five out of six MMEs evaluated in our fuzzing contained missing bounds checks, type confusion and/or off-by-one vulnerabilities specifically within ASN.1 PER decoding routines.

7 DISCUSSION

Responsible Disclosure Efforts. For each vulnerability discovered, we notified the vendors of the respective core implementations of our findings. As part of our disclosures, we included sample crashing inputs for each vulnerability to reproduce the effect and indicated the portion of their source code containing the vulnerability, where applicable. Most vendors have responded to our contact and confirmed the legitimacy of these findings, including all vendors of LTE/5G cores actively used in commercial settings, and many have already incorporated fixes into their projects.

Limitations and Future Work. Our fuzzing efforts are intentionally scoped to the threat model faced by interfaces between the RAN and the cellular core. As we discuss in Section 8, most efforts in cellular fuzzing have focused on either interactions between basebands and base stations or direct UE interactions with the core. Future fuzzing work may be needed to explore interactions among components within the cellular core itself.

While our work improves state coverage by adding protocol shims where needed, a fundamental limitation of our fuzzing approach is that it cannot explore interactions caused as a result of chains of message exchanges. Future work may incorporate additional mechanisms to explore multi-message exchanges between the base station and MM/GW component or include additional setup routines to produce more starting states from which to fuzz.

The underlying coverage-instrumentation tool used for RANsacked was AFLPlusPlus; as such, the fuzzing approach is limited to C/C++ projects. We note that a few open-source 5G implementations (such as free5GC and SD-Core) are implemented in the Go programming language, so adapting this work to such projects would necessitate a change in fuzzing engine and instrumentation.

8 RELATED WORK

Security in Telephone Networks. Since the inception of the Public Switched Telephone Network, security research in telephony has focused on operations that are exposed to public access–whether intentionally or not. One of the earliest examples of such unintentionally exposed functionality is the discovery and exploitation of in-band telephone signalling to circumvent call costs, dubbed "phone phreaking" [48]. Phone networks were subsequently provisioned with an updated, out-of-band signalling protocol named SS7 that mitigated such exploits. As phone networks have evolved more rapidly over the last twenty years, so too have the incidence of security vulnerabilities and corresponding countermeasures. Newer communication methods such as SMS/MMS brought with them security concerns ranging from botnets [53] to denial of service attacks on individual phones [41] and even entire networks [20, 54] that necessitated architectural changes. The introduction of home-use femtocells, followed by more easily-accessible gNodeB base stations in 5G deployments, represent a further shift in security dynamics: where once physically locked-down, RAN equipment is now openly exposed to physical adversarial threats. Our work explores the implications of this final area by enabling performant fuzzing interfaces that have historically been assumed implicitly secure but now face imminent threats.

General Network Fuzzing. The asynchronous, stateful and often concurrent nature of network components create unique challenges that several fuzzing frameworks have attempted to tackle.

AFLNet [45] advanced network fuzzing by enabling packet sending while replacing file-level I/O with socket-based I/O in AFL. However, AFLNet requires a response for every request sent and employs timeouts to handle edge cases, leading to severely degraded performance in fuzzing protocols that may not have a 1-to-1 request/response pattern (as is the case for S1AP/NGAP and GTP). SnapFuzz [4] builds atop AFLNet and leverages fast synchronous network communication instead of slow asynchronous I/Os that greatly accelerates fuzzing speed, though with the same drawbacks of required messages as AFLNet. On the other hand, NyxNet [49] proposed a novel snapshot-based fuzzing framework using hypervisor-based snapshot to ensure the deterministic state of network system before fuzzing with an input. These existing approaches all aim to solve the issue of passing initial fuzzing inputs to the network-attached program while maintaining a consistent initial state, but none of these approaches attempt to account for multi-interfaced systems.

Fuzzing of Cellular Protocols. Advancements in vulnerability discovery have been applied to newer cellular networks in an attempt to cover an increasingly widening attack surface, such as formal modeling of programs and documentation [5, 6, 14, 30, 31, 34, 44] and protocol fuzzing for both cellular devices and select core interfaces. Prior research has explored various ways to fuzz core interfaces accessible directly by UE, such as by leveraging existing NAS testing harnesses (T-Fuzz) [33], performing automata reconstruction of the NAS protocol [15], mutating messages passed between a legitimate UE and eNodeB [24, 47], adding NAS state machine awareness to fuzzing efforts [27], and dynamic semi-automated property testing of UE messages [35]. Such works focus on either the NAS protocol between the UE and MME [15, 27, 33, 35, 47] or the network stack that the UE and eNodeB interface on [24]; most works were evaluated on only one implementation [27, 33, 47]. More recently, extensive work has been done to enable fuzzing of the baseband firmware that resides within cellphones. BaseSAFE [39] applies coverage-guided fuzzing to UE firmware by rehosting individual functions within specific firmware implementations; Firmwire [28] further explores baseband fuzzing by creating a generalizable framework capable of emulating basebands from hundreds of devices. Such works focus exclusively on UE devices rather than core components. Apart from UE-specific fuzzing, HFuzz [38] targets the GTP Control Plane (GTP-C) protocol (distinct from GTP) used for Narrow-Band Internet of Things devices. Our work targets fuzzing from the perspective of a rogue eNodeB/gNodeB or cellphone, which opens up a significantly wider surface area of attack.

9 CONCLUSION

Cellular cores control a wide range of critical functionality. As such, they represent a tempting potential target for attack. To improve the security of these systems, we conduct the first structure-aware fuzzing effort on both open-source and commercial cellular cores. Because the interactions between the nodes in these systems are extremely complex, we incorporate a number of mechanisms to dramatically improve the reach and speed of our campaigns. We additionally develop and release ASNFuzzGen, which facilitates structure-aware fuzzing for arbitrary ASN.1 specifications across many other cellular interfaces. RANsacked discoveres 119 vulnerabilities that enable denial of service and memory corruption attacks. In so doing, we help to advance the state of the art for fuzzing cellular interfaces, and demonstrate that much work remains to be done to ensure a sufficient standard of robustness in LTE/5G core implementations.

ACKNOWLEDGMENTS

We thank Gabriella Neris for her assistance with the cellular testbed. This work was supported in part by the National Science Foundation grants CNS-2054911, CNS-2055014, CNS-1933208, an NSF Graduate Fellowship, and the Air Force Office for Scientific Research grant FA8650-19-1-1969. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] 3GPP. 2018. *3GPP TS 38.331 V15.3.0.* Technical Report. 3GPP. https://www.etsi.org/deliver/etsi_ts/138300_138399/138331/15.03.00_60/ts_138331v150300p.pdf
[2] 3GPP. 2019. *3GPP TS 36.423 V15.5.0.* Technical Report. https://www.etsi.org/deliver/etsi_ts/136400_136499/136423/15.05.00_60/ts_136423v150500p.pdf
[3] 3GPP. 2023. *3GPP TS 24.301 V16.9.0.* Technical Report. ETSI. https://www.3gpp.org/ftp/Specs/archive/24_series/24.301/24301-g90.zip
[4] Anastasios Andronidis and Cristian Cadar. 2022. SnapFuzz: High-Throughput Fuzzing of Network Applications. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis.*
[5] Myrto Arapinis, Loretta Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. 2012. New Privacy Issues in Mobile Telephony: Fix and Verification. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS).*
[6] Myrto Arapinis, Loretta Ilaria Mancini, Eike Ritter, and Mark Dermot Ryan. 2017. Analysis of Privacy in Mobile Telephony Systems. *International Journal of Information Security* 16, 5 (2017), 491–523.
[7] Cornelius Aschermann, Tommaso Frassetto, Thorsten Holz, Patrick Jauernig, Ahmad-Reza Sadeghi, and Daniel Teuchert. 2019. NAUTILUS: Fishing for Deep Bugs with Grammars. *Proceedings of the ISOC Network and Distributed System Security (NDSS) Symposium* (2019).
[8] Cornelius Aschermann, Sergej Schumilo, Tim Blazytko, Robert Gawlik, and Thorsten Holz. 2019. REDQUEEN: Fuzzing with Input-to-State Correspondence. In *NDSS*, Vol. 19. 1–15.
[9] Athonet. [n. d.]. *Our Customers.* https://athonet.com/customers/
[10] AT&T. 2023. *AT&T Cell Booster.* https://web.archive.org/web/20230622145138/https://www.att.com/buy/accessories/Specialty-Items/att-cell-booster.html
[11] Osbert Bastani, Rahul Sharma, Alexander Aiken, and Percy Liang. 2017. Synthesizing Program Input Grammars. *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2017).
[12] Tim Blazytko, Cornelius Aschermann, Moritz Schlögel, Ali Reza Abbasi, Sergej Schumilo, Simon Wörner, and Thorsten Holz. 2019. GRIMOIRE: Synthesizing Structure while Fuzzing. In *USENIX Security Symposium.*
[13] BNAmericas. [n. d.]. *Brisanet Brings Connectivity to Remote Areas in the Northeast with Magma.* https://web.archive.org/web/20240123162926/https://www.bnamericas.com/en/news/brisanet-brings-connectivity-to-remote-areas-in-the-northeast-with-magma
[14] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. 2021. Bookworm Game: Automatic Discovery of LTE Vulnerabilities through Documentation Analysis. In *2021 IEEE Symposium on Security and Privacy (SP).* IEEE, 1197–1214.
[15] Merlin Chlosta, David Rupprecht, and Thorsten Holz. 2021. On the Challenges of Automata Reconstruction in LTE Networks. *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (2021).
[16] Sprint Corporation. 2023. *Sprint Airave Support.* https://web.archive.org/web/20230622145651/https://www.sprint.com/en/support/solutions/device/airave-support-center.html
[17] Doug DePerry, Tom Ritter, and Andrew Rahimi. 2013. *Traffic Interception & Remote Mobile Phone Cloning with a Compromised CDMA Femtocell.* Technical Report. DEF CON.
[18] NTT DoCoMo. 2024. *Super-Compact Base Station for Femtocells.* https://web.archive.org/web/20240122234628/https://www.docomo.ne.jp/english/corporate/technology/rd/tech/network/femtocells/
[19] DrmnSamoLiu. 2018. *CVE-2018-6311/CVE-2018-6312.* https://gist.github.com/DrmnSamoLiu/cd1d6fa59501f161616686296aa4a6c8

[20] William Enck, Patrick Traynor, Patrick McDaniel, and Thomas La Porta. 2005. Exploiting Open Functionality in SMS-Capable Cellular Networks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[21] ENISA. 2021. *Telecom Security Incidents 2020 - Annual Report*. Technical Report. European Union Agency for Cybersecurity. https://www.enisa.europa.eu/publications/telecom-annual-incident-reporting-2020

[22] ETSI. 2024. *ETSI EN 302 637-2 V1.4.1*. Technical Report. https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.04.01_60/en_30263702v010401p.pdf

[23] Facebook. 2023. *Folly: Facebook Open-Source Library*. Facebook. https://github.com/facebook/folly

[24] Matheus E. Garbelini, Zewen Shang, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. 2022. Towards Automated Fuzzing of 4G/5G Protocol Implementations Over the Air. *IEEE Global Communications Conference (GLOBECOM)* (2022), 86–92.

[25] O2 Germany. 2023. *O2 Business Femtocell/Signal Box*. https://web.archive.org/web/20240122235101/https://www.businesstarife.de/femtocell/

[26] N Golde, R Borgaonlar, and K Redon. 2011. *Femtocells: Poisonous Needle in the Operator's Hay Stack*. Technical Report. Blackhat USA.

[27] Fengjiao He, Wenchuan Yang, Baojiang Cui, and Jia Cui. 2022. Intelligent Fuzzing Algorithm for 5G NAS Protocol Based on Predefined Rules. *International Conference on Computer Communications and Networks (ICCCN)* (2022), 1–7.

[28] Grant Hernandez, Marius Muench, Dominik Maier, Alyssa Milburn, Shinjo Park, Tobias Scharnowski, Tyler Tucker, Patrick Traynor, and Kevin Butler. 2022. FIRMWIRE: Transparent Dynamic Analysis for Cellular Baseband Firmware. *Proceedings of the ISOC Network and Distributed Systems Security (NDSS) Symposium* (2022).

[29] Matthias Hoschele and Andreas Zeller. 2017. Mining Input Grammars with AUTOGRAM. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 31–34.

[30] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. 2018. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In *Proceedings of the ISOC Network and Distributed System Security (NDSS) Symposium*.

[31] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 2019. 5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[32] Jio. 2024. *JioConnect*. https://www.jio.com/business/jio-connect

[33] William Johansson, Martin Svensson, Ulf Larson, Magnus Almgren, and Vincenzo Gulisano. 2014. T-Fuzz: Model-based Fuzzing for Robustness Testing of Telecommunication Protocols. *Seventh International Conference on Software Testing, Verification and Validation* (2014), 323–332.

[34] Eunsoo Kim, Dongkwan Kim, CheolJun Park, Insu Yun, and Yongdae Kim. 2021. BaseSpec: Comparative Analysis of Baseband Software and Cellular Specifications for L3 Protocols. In *Proceedings of the ISOC Network and Distributed System Security (NDSS) Symposium*.

[35] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. 2019. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. *IEEE Symposium on Security and Privacy (SP)* (2019), 1153–1168.

[36] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. 2018. Evaluating Fuzz Testing. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[37] Sukchan Lee. 2023. *Support - Open5GS*. https://web.archive.org/web/20230930053106/https://open5gs.org/open5gs/support/

[38] Xinyao Liu, Baojiang Cui, Junsong Fu, and Jinxin Ma. 2020. HFuzz: Towards Automatic Fuzzing Testing of NB-IoT Core Network Protocols Implementations. *Future Generation Computer Systems* 108 (2020), 390–400.

[39] Dominik Maier, Lukas Seidel, and Shinjo Park. 2020. BaseSAFE: Baseband Sanitized Fuzzing Through Emulation. *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (2020), 122–132.

[40] Gabi Melman. 2023. *Spdlog: Fast C++ logging library*. Github. https://github.com/gabime/spdlog

[41] Collin Mulliner, Nico Golde, and Jean-Pierre Seifert. 2011. SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale. In *Proceedings of the USENIX Security Symposium*.

[42] O2. 2023. *Boostbox Guide*. https://web.archive.org/web/20231010170442/https://www.o2.co.uk/help/network-coverage-and-international/boostbox-guide

[43] OpenAirInterface. [n. d.]. *OSA Members - OpenAirInterface*. https://web.archive.org/web/20231226212507/https://openairinterface.org/osa-members/

[44] CheolJun Park, Sangwook Bae, BeomSeok Oh, Jiho Lee, Eunkyu Lee, Insu Yun, and Yongdae Kim. 2021. DoLTEst: In-depth Downlink Negative Testing Framework for LTE Devices. In *Proceedings of the USENIX Security Symposium*.

[45] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. 2020. AFLNet: A Greybox Fuzzer for Network Protocols. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 460–465.

[46] Van-Thuan Pham, Marcel Böhme, Andrew E Santosa, Alexandru Răzvan Căciulescu, and Abhik Roychoudhury. 2019. Smart Greybox Fuzzing. *IEEE Transactions on Software Engineering* 47, 9 (2019), 1980–1997.

[47] Srinath Potnuru and Prajwol Kumar Nakarmi. 2021. Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G. *2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (2021), 295–300.

[48] Ron Rosenbaum. 1971. Secrets of the Little Blue Box. *Esquire Magazine* 76 (1971), 117–125.

[49] Sergej Schumilo, Cornelius Aschermann, Andrea Jemmett, Ali Reza Abbasi, and Thorsten Holz. 2022. Nyx-Net: Network Fuzzing with Incremental Snapshots. *Proceedings of the Seventeenth European Conference on Computer Systems* (2022).

[50] Toby Shapshak. [n. d.]. *African Internet Connectivity Gets a Mobile World Congress Boost*. https://www.forbes.com/sites/tobyshapshak/2019/02/27/african-internet-connectivity-gets-a-mobile-world-congress-boost/?sh=434ed2c4c607

[51] European Space Agency. 2015. *ASN1SCC - ASN.1 Space Certifiable Compiler*. https://essr.esa.int/project/asn1scc-asn-1-space-certifiable-compiler

[52] T-Mobile. 2023. *Register a Signal Booster*. https://web.archive.org/web/20230622145341/https://www.t-mobile.com/support/coverage/register-a-signal-booster

[53] Patrick Traynor, Michael Lin, Machigar Ongtang, Vikhyath Rao, Trent Jaeger, Patrick McDaniel, and Thomas La Porta. 2009. On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[54] Patrick Traynor, Patrick McDaniel, and Thomas La Porta. 2007. On Attack Causality in Internet-Connected Cellular Networks. In *Proceedings of the USENIX Security Symposium*.

[55] Patrick Traynor, Patrick McDaniel, and Thomas La Porta. 2008. *Security for Telecommunications Networks*. Vol. 40. Springer Science & Business Media.

[56] Verizon. 2023. *Verizon LTE Network Extender*. https://web.archive.org/web/20230622145025/https://www.verizon.com/products/verizon-lte-network-extender/

[57] J Webb-Twoomey. 2023. *The Rising Threat Landscape for Cell Towers*. BioConnect. https://bioconnect.com/2023/05/31/the-rising-threat-landscape-for-cell-towers/

[58] Y Zheng and H Shan. 2015. *Hacking Femtocell*. Technical Report. DEF CON.

A ABBREVIATED TERMS

3GPP 3rd Generation Partnership Project
5G 5th-Generation Cellular Network
AFL American Fuzzy Lop
AGW Access Gateway
AMF Access and Mobility Management Function
ASN.1 Abstract Syntax Notation No. 1
EMM EPS Mobility Management
ESM EPS Session Management
EPC Evolved Packet Core
EPS Evolved Packet System
GPRS General Packet Radio Service
GSM Global System for Mobile Communications
GTP GPRS Tunnelling Protocol
GTP-C GTP Control Plane
GW Gateway component
IE Information Element
LTE Long-Term Evolution (4G)
MM Mobility Management component
MME Mobility Management Entity
NAS Non-Access Stratum
NGAP Next Generation Application Protocol
PCR Persistent mode with Channel Reset
PER Packed Encoding Rules
RAN Radio Access Network
S1AP S1 Application Protocol
SCTP Stream Control Transmission Protocol
SGW Serving Gateway
UE User Equipment (cellular mobile devices)

Table 6: Vulnerabilities Discovered in RANsacked

Cellular Core	*Proto*	*VULN ID*	*CVE*	*Vulnerability*	*General Cause*	*Threat*
Open5GS (5G)	NAS	VULN-A01	CVE-UNASSIGNED	Assertion	Malformed Field	DOS
		VULN-A02	CVE-2024-24428	Assertion	Payload Size	DOS
		VULN-A04	CVE-2024-24427	Assertion	Malformed Field	DOS
	NGAP	VULN-A03	CVE-UNASSIGNED	Assertion	Malformed Field	DOS
		VULN-A05	CVE-UNASSIGNED	Assertion	Malformed Field	DOS
OAI (5G)	NGAP	VULN-B01	CVE-2024-24442	Null Dereference	Invalid Packet Type	DOS
		VULN-B02	CVE-DUPLICATE	Buffer Overflow	Missing Field	Mem. Corrupt
		VULN-B03	CVE-UNASSIGNED	Uninitialized Pointer	Missing Field	Mem. Corrupt
		VULN-B04	CVE-2024-24447	Buffer Overflow	Packet Size	RCE
		VULN-B05	CVE-2024-24451	Buffer Overflow	FD_SET Overflow	Mem. Corrupt
		VULN-B06	CVE-2024-24444	FD Exhaustion	FD Exhaustion	DOS
		VULN-B07	CVE-DUPLICATE	Null Dereference	ASN.1 Decode Failure	DOS
		VULN-B08	CVE-2024-24449	Uninitialized Pointer	Missing Field	Mem. Corrupt
		VULN-B09	CVE-2024-24446	Uninitialized Pointer	Missing Field	Mem. Corrupt
		VULN-B10	CVE-2024-24443	Uninitialized Pointer	Malformed Field	Mem. Corrupt
		VULN-B11	CVE-2024-24447	Uninitialized Pointer	Malformed Field	Mem. Corrupt
Magma (5G)	NAS	VULN-C01	CVE-2024-24425	Buffer Overflow	Payload Size	Mem. Corrupt
	NGAP	VULN-C02	CVE-2024-24426	Assertion	Missing Field	DOS
		VULN-C03	-	Assertion	Missing Field	DOS
		VULN-C04	-	Assertion	Missing Field	DOS
		VULN-C05	-	Assertion	Missing Field	DOS
		VULN-C06	-	Assertion	Missing Field	DOS
		VULN-C07	-	Assertion	Missing Field	DOS
		VULN-C08	-	Assertion	Missing Field	DOS
		VULN-C09	-	Assertion	Missing Field	DOS
		VULN-C10	-	Assertion	Missing Field	DOS
		VULN-C11	-	Assertion	Missing Field	DOS
		VULN-C12	-	Assertion	Missing Field	DOS
		VULN-C13	-	Assertion	Missing Field	DOS
		VULN-C14	-	Assertion	Missing Field	DOS
		VULN-C15	-	Assertion	Missing Field	DOS
		VULN-C16	-	Assertion	Missing Field	DOS
		VULN-C17	-	Assertion	Missing Field	DOS
		VULN-C18	-	Assertion	Missing Field	DOS
		VULN-C19	-	Assertion	Missing Field	DOS
		VULN-C20	-	Assertion	Missing Field	DOS
Magma/OAI (LTE)	NAS	VULN-D01	CVE-2023-37024	Assertion	Unimplemented	DOS
		VULN-D06	CVE-2023-37029	Assertion	Packet Size	DOS
		VULN-D09	CVE-2023-37032	Buffer Overflow	Packet Size	Mem. Corrupt
		VULN-D16	CVE-2024-24419	Int. Overflow	Malformed Field	Mem. Corrupt
		VULN-D17	CVE-2024-24416	Int. Overflow	Malformed Field	Mem. Corrupt
		VULN-D18	CVE-2024-24424	Assertion	Malformed Field	DOS
		VULN-D19	CVE-2024-24420	Assertion	Unimplemented	DOS
		VULN-D20	CVE-2024-24418	Int. Overflow	Malformed Field	Mem. Corrupt
		VULN-D21	CVE-2024-24421	Type Confusion	Header Mismatch	Mem. Corrupt
		VULN-D22	CVE-2024-24423	Int. Overflow	Malformed Field	Mem. Corrupt
		VULN-D23	CVE-2024-24417	Buffer Overflow	Malformed Field	DOS
		VULN-D24	CVE-2024-24422	Buffer Overflow	Packet Size	Mem. Corrupt
		VULN-D25	CVE-2023-37039	Null Dereference	Malformed Field	DOS
	NGAP	VULN-D02	CVE-2023-37025	Null Dereference	Missing Field	DOS

Continued on next page

Table 6: Vulnerabilities Discovered in RANsacked (Continued)

Cellular Core	*Proto*	*VULN ID*	*CVE*	*Vulnerability*	*General Cause*	*Threat*
Magma/OAI (LTE)	S1AP	VULN-D03	CVE-2023-37026	Null Dereference	Missing Field	DOS
		VULN-D04	CVE-2023-37027	Null Dereference	Missing Field	DOS
		VULN-D05	CVE-2023-37028	Null Dereference	Missing Field	DOS
		VULN-D07	CVE-2023-37030	Null Dereference	Missing Field	DOS
		VULN-D08	CVE-2023-37031	Null Dereference	Missing Field	DOS
		VULN-D10	CVE-2023-37033	Null Dereference	Missing Field	DOS
		VULN-D11	CVE-2023-37034	Null Dereference	Missing Field	DOS
		VULN-D12	CVE-2023-37035	Null Dereference	Missing Field	DOS
		VULN-D13	CVE-2023-37036	Null Dereference	Missing Field	DOS
		VULN-D14	CVE-2023-37037	Null Dereference	Missing Field	DOS
		VULN-D15	CVE-2023-37038	Null Dereference	Missing Field	DOS
NextEPC (LTE)	S1AP	VULN-E01	CVE-2023-36997	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-E02	CVE-2023-37000	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-E03	CVE-2024-24438	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-E04	CVE-2024-24439	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-E05	CVE-2024-24440	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-E06	CVE-2024-24441	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-E07	CVE-2024-24437	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
	NAS	VULN-E08	CVE-2023-36998	No Bounds Check	Malformed NAS Field	Mem. Corrupt
		VULN-E09	CVE-2023-36999	No Bounds Check	Malformed NAS Field	Mem. Corrupt
Open5GS (LTE)	S1AP	VULN-F01	CVE-2023-37002	Assertion	Missing Field	DOS
		VULN-F02	CVE-2023-37003	Assertion	Missing Field	DOS
		VULN-F03	CVE-2023-37004	Assertion	Missing Field	DOS
		VULN-F04	CVE-2023-37005	Assertion	Missing Field	DOS
		VULN-F05	CVE-2023-37006	Assertion	Missing Field	DOS
		VULN-F06	CVE-2023-37007	Assertion	Missing Field	DOS
		VULN-F07	CVE-2023-37008	Off-By-One	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-F08	CVE-2023-37009	Assertion	Missing ASN.1 Field	DOS
		VULN-F09	CVE-2023-37010	Assertion	Missing ASN.1 Field	DOS
		VULN-F10	CVE-2023-37011	Assertion	Missing ASN.1 Field	DOS
		VULN-F11	CVE-2023-37012	Assertion	Missing ASN.1 Field	DOS
		VULN-F12	CVE-2023-37013	Assertion	Protocol State	DOS
		VULN-F13	CVE-2023-37014	Assertion	Missing Field	DOS
		VULN-F14	CVE-2023-37015	Assertion	Missing Field	DOS
		VULN-F15	CVE-2023-37016	Assertion	Missing Field	DOS
		VULN-F16	CVE-2023-37017	Assertion	Missing Field	DOS
		VULN-F17	CVE-2023-37018	Assertion	Missing Field	DOS
		VULN-F18	CVE-2023-37019	Assertion	Missing Field	DOS
		VULN-F19	CVE-2023-37020	Assertion	Missing Field	DOS
		VULN-F20	CVE-2023-37021	Assertion	Missing Field	DOS
		VULN-F21	CVE-2023-37022	Assertion	Missing Field	DOS
		VULN-F22	CVE-2023-37023	Assertion	Missing Field	DOS
		VULN-F23	-	Assertion	Missing Field	DOS
	NAS	VULN-F24	CVE-2024-24431	Assertion	Unexpected Size	DOS
		VULN-F25	CVE-2024-24429	Assertion	Malformed Field	DOS
		VULN-F26	CVE-2024-24430	Assertion	Malformed Field	DOS
		VULN-F27	CVE-2024-24432	Assertion	Packet Size	DOS
	GTP	VULN-F28	-	Assertion	Protocol State	DOS
		VULN-F29	-	Assertion	Protocol State	DOS

Continued on next page

Table 6: Vulnerabilities Discovered in RANsacked (Continued)

Cellular Core	*Proto*	*VULN ID*	*CVE*	*Vulnerability*	*General Cause*	*Threat*
srsEPC (LTE)	S1AP	VULN-G01	CVE-2023-37001	No Bounds Check	Faulty ASN.1 Codegen	Mem. Corrupt
SD-Core (LTE)	S1AP	VULN-H01	CVE-2023-37040	Type Confusion	Faulty ASN.1 Codegen	Mem. Corrupt
		VULN-H02	CVE-2023-37042	Off-By-One	Memory Init.	Mem. Corrupt
		VULN-H03	CVE-2024-24436	Buffer Overflow	Malformed Field	Mem. Corrupt
		VULN-H04	CVE-2024-24435	Buffer Overflow	Malformed Field	Mem. Corrupt
		VULN-H05	CVE-2024-24433	Buffer Overflow	Malformed Field	Mem. Corrupt
		VULN-H06	CVE-2024-24434	Buffer Overflow	Malformed Field	Mem. Corrupt
		VULN-H07	CVE-2023-37041	Buffer Overflow	Malformed Field	Mem. Corrupt
	NAS	VULN-H08	CVE-2023-37043	Buffer Overflow	Payload Size	Mem. Corrupt
		VULN-H09	CVE-2023-37044	Buffer Overflow	Payload Size	Mem. Corrupt
Athonet (LTE)	S1AP	VULN-I01	CVE-2024-24454	Unknown	Faulty ASN.1 Codegen[a]	DOS[b]
		VULN-I02	CVE-2024-24459	Unknown	Faulty ASN.1 Codegen[a]	DOS[b]
		VULN-I03	CVE-2024-24455	Unknown	Protocol State	DOS[b]
		VULN-I04	CVE-2024-24457	Unknown	Faulty ASN.1 Codegen[a]	DOS[b]
		VULN-I05	CVE-2024-24452	Unknown	Faulty ASN.1 Codegen[a]	DOS[b]
		VULN-I06	CVE-2024-24453	Unknown	Faulty ASN.1 Codegen[a]	DOS[b]
		VULN-I07	CVE-2024-24458	Unknown	Protocol State	DOS[b]
	NAS	VULN-I08	CVE-2024-24456	Unknown	Malformed Field	DOS[b]
OAI (LTE)	GTP	VULN-J01	-	Assertion	Protocol State	DOS
		VULN-J02	-	Assertion	Packet Size	DOS

[a] Inferred based on crashing/noncrashing behavior associated with certain ASN.1 fields–select base message types containing invalid ProtocolIE-ID fields would crash the Athonet MME. As this field is normally parsed by ASN.1 decoding routines, and all other tested implementations use ASN.1 compilers to generate packet decoding routines, we deemed 'Faulty ASN.1 Codegen' to be the likely cause of crash.

[b] Some Athonet vulnerabilities may also be exploitable memory corruption. Due to a lack of source code or binary access, we were unable to determine root causes of these vulnerabilities beyond packet field/information element characteristics.